REMARKS

Claims 1-35 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 102(b) Rejection:

The Examiner rejected claims 1, 2, 11-15 and 24-26 under 35 U.S.C. § 102(b) as being anticipated by Rotenberg, et al. (hereinafter "Rotenberg"). Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 1, contrary to the Examiner's assertion, Rotenburg does not disclose *wherein the prefetch unit is configured to check the trace cache for a match for the predicted target address in response to the branch prediction unit outputting a predicted target address*; and *wherein the prefetch unit is configured to not check the trace cache for a match until the branch prediction unit outputs the predicted target address*. In the Response to Arguments section of the Office Action mailed February 6, 2007, the Examiner submitted, "It is inherent that you cannot check for a value if you do not know what the value is. The trace cache as disclosed by Rotenberg cannot search for the predicted target address in the cache if it does not know what the predicted target address is, and in fact, no cache or any other type of hardware can find something when it doesn't know what it is looking for, thus, the value must have been output from the branch prediction unit prior to the checking."

Applicants asserted that the Examiner's interpretation of the operation of Rotenberg was incorrect and that he has also misread the specific limitations recited in Applicants' claim. For example, in the system of Rotenberg, the trace cache is checked for a match for every fetch address, not only for target addresses generated by the branch prediction unit and BTB logic. As stated on p. 28, second column, beginning of the first full paragraph, "The trace cache is accessed in parallel with the instruction cache and BTB using the current fetch address… The fetch address is used together with the

multiple branch predictions to determine if the trace read from the trace cache matches the predicted sequence of basic blocks. Specifically, a trace cache hit requires that (1) the fetch address matches the tag and (2) the branch predictions match the branch flags… On a trace cache hit, an entire trace of instructions is fed into the instruction latch, bypassing the instruction cache" (emphasis added.) The next paragraph goes on to state, "On a trace cache miss, fetching proceeds normally from the instruction cache, i.e., contiguous instruction fetching." **It is clear from these passages that every fetch address is compared to the entries in the trace cache, regardless of the operation of the branch prediction unit.**

As described in Applicants' Response of October 23, 2006, Rotenberg's trace cache operation is clearly different from Applicants' claimed invention, in that it teaches searching for a hit in the trace cache first (i.e., on every cycle), and then fetching from the instruction cache if there is not a hit. Applicants' claim 1 recites *wherein the prefetch unit is configured to not check the trace cache for a match until the branch prediction unit outputs the predicted target address*. **In other words, the trace cache is not checked for a match on every cycle. Instead, it is only checked for a match on cycles for which the branch prediction unit outputs a predicted target address.** Therefore, Applicants again assert that Rotenberg clearly does not anticipate claim 1.

In the Response to Arguments section of the Final Action mailed July 18, 2007, the Examiner submits, "When looking at the entire claim in context, it can be seen that the claim is referring to searching the cache for a match for the predicted target address, and that it cannot search the cache for the predicted target address until the predicted target address is generated" (emphasis Examiner's). **The Examiner is incorrect.** The referenced limitation of the claim does not recite "not check the trace cache for the match" (i.e., referring to the match for the predicted target address of the previous limitation), but instead recites "not check the trace cache for a match" (i.e., any match between the current address and the trace cache). Thus, the plain language of the claim does not support the Examiner's interpretation. Applicants also assert that one of

ordinary skill in the art having benefit of Applicants' disclosure could not make this interpretation of the referenced limitation.

As explicitly recited in claim 1, multiple instructions are fetched from the instruction cache, **without checking for a hit in the trace cache**, until the branch prediction unit outputs a predicted branch address (i.e., until a branch instruction is encountered for which the branch prediction unit outputs a predicted target address). Only then (in response to this output) is the trace cache checked for a trace cache hit. By contrast, in Rotenberg, the instruction cache and trace cache are accessed in parallel on every cycle (i.e., for each instruction fetch, not after fetching multiple instructions from the instruction cache, as recited in claim 1), regardless of whether there is a branch instruction for which a predicted target address is output. In other words, in Rotenberg, checking the trace cache for a hit is not limited to cycles for which a branch prediction unit outputs a predicted target address, as required by Applicants' claim. **The Examiner's repeated assertions that it is impossible to search for something that has not yet been generated are irrelevant to Applicants' argument and to what is actually recited in the claim.** In the system of Rotenberg, the trace cache is "searched" (i.e., checked for a cache hit) on every cycle, regardless of whether there is a predicted branch target address output (i.e., regardless of whether such a predicted target address has been generated). **In contrast**, claim 1 explicitly recites that multiple instructions are fetched from the instruction cache, without checking for a hit in the trace cache, until the branch prediction unit outputs a predicted branch address.

For at least the reasons above, the rejection of claim 1 is unsupported by the cited art and removal thereof is respectfully requested. Claim 14 includes limitations similar to claim 1, and so the arguments presented above apply with equal force to this claim, as well.

Section 103(a) Rejections:

The Examiner rejected claims 3 and 16 under 35 U.S.C. § 103(a) as being unpatentable over Rotenberg in view of Patterson, et al. (hereinafter "Patterson"), claims 4, 10, 17 and 23 as being unpatentable over Rotenberg in view of Braught, and further in view of Xia, claims 5-8 and 18-21 as being unpatentable over Rotenberg, Xia, Braught and further in view of Lange, et al. (U.S. Patent 3,896,419) (hereinafter "Lange"), claims 9 and 22 as being unpatentable over Rotenberg, Xia and Braught in view of Akkary, et al. (U.S. Patent 6,247,121) (hereinafter "Akkary"), claims 27-31 and 35 as being unpatentable over Rotenberg, Xia, Braught and Lange in view of Akkary, and claims 32 and 34 as being unpatentable over Rotenberg in view of Xia. Applicants respectfully traverse this rejection for at least the following reasons.

Regarding claim 27, contrary to the Examiner's assertion, the cited references fail to teach or suggest all the limitations of claim 27. As discussed at length in Applicants' Response of October 23, 2006, the cited references fails to teach or suggest *receiving a retired instruction*. The Examiner admits that Rotenberg does not teach that instructions need to be retired before the trace can be generated and relies on Akkary to teach this limitation. The Examiner submits that Akkary teaches that instructions are not put into the trace buffers until they have been retired, to ensure that they executed correctly (column 3, lines 40-44). As noted in Applicants' previous Responses, the Examiner has incorrectly interpreted this passage as teaching that instructions are not put into the trace buffers until they have been retired. **This passage actually says just the opposite**, that is, that instructions placed in the trace buffer stay in the trace buffer until they are retired.

The Examiner previously argued, "Therefore, Akkary suggests the limitation of using a retired instruction, by providing one of ordinary skill in the art with a motivation for using a retired instruction, as opposed to in-flight instructions." In the Response to Arguments section of the Office Action of February 6, 2007, the Examiner disagrees with Applicants' argument that Akkary does not teach using a retired instruction for any particular purpose (much less for building traces) and that the motivation of "incorrect"

traces is not valid. The Examiner again includes remarks regarding the correctness of instruction execution, and submits, "The idea behind Examiners statement of an "incorrect" trace is a trace which does not represent the flow of the instruction stream, that is, it does not represent the series of branches that occurred in correct execution of the program, but rather, an alternate series of instructions that did not occur. While this data may be valuable in a trace cache, as it will provide the correct path in the event that the path will be a correct execution in the future, it does not represent what did happen, and Rotenberg's motivation for creating the trace cache was to exploit code reuse, both points being given in Section 1.1, that an instruction which was used recently will be used again in the future, and that branches tend to be biased in one directions, and it is likely that certain paths will be followed frequently." While these goals for Rotenberg's invention are present, this section also explicitly states that the trace line is filled as instructions are fetched from the instruction cache, clearly teaching away from filling the trace line with retired instructions.

The Examiner also points to Rotenberg, Section 2.3, which describes that some traces are committed but never used, thus evicting a useful trace. However, this passage (point 6 "judicious trace selection") describes that a way to avoid this situation is to include an additional small buffer to store recent traces and that the traces in this buffer may only be committed to the trace cache after one or more hits to the trace. This clearly does not suggest waiting until an instruction was retired before considering adding it to the trace, as it refers to a way to avoid a completely unused trace, not an un-retired individual instruction. Similarly, the Examiner's additional reference to point 5 of Section 2.3 ("fill issues") refers to whether or not to fill the trace cache with speculative traces (i.e., constructed traces), not to whether to add an individual un-retired instruction to a trace or to start construction of a trace using an un-retired instruction.

In Response to the above argument, the Examiner (in the Final Action mailed July 18, 2007) submits, "Applicant has ignored the rejection in the context of which it was given, and is misreading Rotenberg in addition to this. Rotenberg suggests one possible solution, it is never stated that it is the only or even the best way to deal with this

problem, but as Examiner pointed out in the rejection, the key feature is that there is clearly a very damaging effect on the trace cache for storing non-useful traces." The Examiner goes on to suggest that Rotenberg suggests a motivation to wait for a correct result for the execution (by the branch outcome being resolved), that Akkary clearly states that a retired instruction is guaranteed to have executed correctly, and that together this is more than enough motivation to combine the references. **Applicants again assert, however, that combining the references still does not teach the claimed invention,** as neither reference teaches that a solution to this problem is to receive retired instructions, and in response to determining that a previous trace under construction duplicates a trace in a trace cache and that the received instruction corresponds to a branch label, **beginning construction of a new trace.** The fact that the system of Rotenberg could solve this problem in a manner different from the only example described does not suggest this specific solution. In addition, the teachings of Akkary do not overcome Rotenberg in teaching the claimed invention, since Akkary does not teach starting a trace cache in response to determining (among other things) that a received retired instruction corresponds to a branch label. **Instead, Akkary's solution explicitly teaches that instructions are put into a trace cache prior to retirement and that they stay there until they are retired.**

Applicants further asserted that the cited references fail to teach or suggest *in response to determining that a previous trace under construction duplicates a trace in a trace cache and that the received instruction corresponds to a branch label, beginning construction of a new trace.*

The Examiner admitted that Rotenberg fails to teach *if a previous trace under construction duplicates a trace in a trace cache, delaying construction of the new trace until the received instruction corresponds to a branch label.* The Examiner admits that Rotenberg does not discuss the issue of duplication, but discusses the disadvantages which occur when a trace cache miss occurs while servicing a previous trace cache miss, and teaches the disadvantages of a useless trace displacing a useful trace. The Examiner then submit, "Therefore, Rotenberg teaches a system in which allows tracing of potential

duplicate traces, and also a system which requires action when a miss occurs while servicing a miss." Applicants assert that the Examiner is contradicting himself and that he has cited nothing in Rotenberg that teaches tracing of potential duplicate traces. Applicants again note that, in the system of Rotenberg, the fill-line buffer logic services trace cache misses (Section 2.2, fourth paragraph). That is, the fill-line buffer fetches instructions because the combination of a matching target address and matching branch flags is not found in the trace cache. Therefore, there is no reason to believe there would ever be a duplicate trace in the system of Rotenberg. The performance problem described in Section 2.3, point 5, therefore, would not be solved by checking for duplicate data before building a trace. Since duplicate traces should not be contained in the trace cache of Rotenberg, there would be no opportunity to avoid such duplication.

The Examiner suggests that there is a "potential for duplicate traces to exist with path associativity in Rotenberg's alternative embodiments." **This is completely unsupported in Rotenberg. Similarly, the Examiner's contention that "Rotenberg indicates in his judicial trace selection that storing a duplicate trace would be at best useless, and at worst displace a useful trace that may be used" is also completely unsupported. No such mention of duplicate traces exists in Rotenberg.**

In the Response to Arguments section of the Final Action mailed July 18, 2007, the Examiner submits, that in the "path associativity" alternate embodiment, "Rotenberg teaches that it may be advantageous to store multiple paths emanating from a given address, as it is possible that a branch in the middle of a trace will create two possible outcomes that should be saved." The Examiner goes on to assert, "Now, it should be very apparent from this teaching that the only possible way to implement this is to perform a trace whenever a trace would normally be started, even on a cache hit, because the path may diverge from the stored trace, and be unique, and there is absolutely no way to know until the trace is complete. However, if it is not unique, then it would end up being a duplicate of the trace that was hit in the cache, and it would be wasteful to allow this trace to enter the cache (where it may overwrite a different trace based on the implementation of the cache). Thus Rotenberg discusses an alternative embodiment in

which duplicates could exist, even though he does not explicitly mention it or say how to deal with it, but one of ordinary skill in the art would be able to understand this by looking at the alternative embodiment." **The Examiner's remarks are completely speculative and do nothing to overcome the deficiencies of the cited references in teaching the specific limitations of the claimed invention.** The Examiner's assertion that "the only possible way to implement" this path associativity involves changing the operation of the system so that it starts a trace even on a cache hit clearly does not teach the above-referenced limitations the claimed invention, which recite *in response to determining that a previous trace under construction duplicates a trace in a trace cache and that the received instruction corresponds to a branch label, beginning construction of a new trace.* The Examiner's suggested "only possible way to implement" the described path associativity of Rotenberg has nothing to do with determining that a previous trace under construction duplicates a trace in a trace cache before beginning construction of a new trace, nor does it involve determining that a received retired instruction corresponds to a branch label.

The Examiner previously submitted that Lange teaches, "a system in which a cache is checked for a value while accessing memory, and if the identical value is found in the cache, the fetch to memory is aborted, freeing the memory to be used by a different operation." Applicants asserted that this description of the operation of a data cache during fetching from main memory has absolutely nothing to do with the limitations of the present invention, as discussed in Applicants' previous Responses. Even if the applications of a conventional cache and a trace cache were analogous, **checking Rotenberg's trace cache for a duplicate trace before collecting a new trace would not teach the specific limitations of claim 27, and is contradictory to the Examiner's remarks above regarding the advantages of including duplicate traces** (i.e., in order to support multiple paths emanating from the same beginning address). **In other words, the Examiner first argues that Rotenberg teaches duplicate traces may exist, and then argues that the combined references teach that the system of Rotenberg should not allow construction of duplicate traces. These arguments cannot coexist if the references are to teach the limitations of claim 27. If no duplicate traces can be**

constructed, there would be no reason to determine if a trace previously under construction was, in fact, duplicating a trace already in the trace cache. Therefore, the Examiner's reasons for combining the references are clearly not supported by the cited art, and the references, when combined, do not teach the limitations recited in claim 27.

The Examiner also previously submitted that in an alternative embodiment, Rotenberg teaches that it is disadvantageous to displace a useful trace with a useless trace and that in his judicious trace selection section, he provides a motivation to prevent these duplicate traces from entering the trace cache from the line fill buffer. This section goes on to describe a solution in which a small buffer stores recent traces, which are not committed to the trace cache until after one or more hits to that trace. **Again, this solution is completely different from the specific limitations recited in claim 27, and nothing in the cited references teaches these specific limitations.** For example, this solution has nothing to do with receiving a retired instruction, which is used to determine whether to start a new branch construction, based on its correspondence with a branch label. Instead, committing a trace to the trace cache depends on getting "one or more hits" to a trace in the buffer. The Examiner submits that this section provides motivation to handle cases involving duplicate traces, and that Lange provides such a method, by simply discarding the duplicate value. **Again, the Examiner's remarks do not address the limitations recited in claim 27, which have nothing to do with discarding a duplicate trace, but with specific conditions to be met before beginning construction of a new trace, which the cited references do not teach.**

For at least the reasons above, the rejection of claim 27 is unsupported by the cited art and removal thereof is respectfully requested. Claim 35 includes limitations similar to claim 27, and so the arguments presented above apply with equal force to this claim, as well.

Regarding claim 32, contrary to the Examiner's assertion, the cited references fail to teach or suggest *a method, comprising: fetching instructions from an instruction*

cache; continuing to fetch instructions from the instruction cache without searching a trace cache until a branch target address is generated; in response to a branch target address being generated, searching a trace cache for an entry corresponding to the branch target address. The Examiner submits that Rotenberg teaches all of the limitations of this claim except, "without searching a trace cache". However, as discussed above regarding claims 1 and 14, Rotenberg does not teach searching a trace cache for an entry corresponding to a branch target address in response to a branch target address being generated. **In fact, the Examiner admits that Rotenberg teaches that the trace cache is searched on every instruction.**

The Examiner submits that Xia teaches an advantage of starting a trace only at predictable branch instructions so that less cache space is required. The Examiner submits, "An effect of only having traces start on branches is that it is then inherent that a non-branch instruction can never be the start of the trace, therefore there would be no reason to search the trace cache for a non-branch instruction, which has further potential advantages such as power saving and potential critical path reduction, as one of ordinary skill in the art would recognize."

Applicants note, however, that in the system taught by Xia, the trace table and instruction cache are checked in parallel on every instruction, just as in the system of Rotenberg. (See Xia, section 3.5, first paragraph.) **Therefore, the Examiner's contention that it would be obvious not to search the trace cache for a non-branch instruction is completely unsupported, as his own references check the trace cache on every instruction. Furthermore, his assertion that such a technique would contribute to power saving or potential critical path reduction is similarly unsupported in the cited art.** In fact, the only advantage noted for starting traces on branches is to save memory space. It is clear from the system of Xia that this memory space can be saved without any change to the method for searching the trace cache on every cycle. **Therefore, there clearly is nothing inherent about such a change in a system that starts traces on branches.**

In the Response to Arguments section of the Final Action mailed July 18, 2007, the Examiner submits that Applicants are misinterpreting Xia and that he believes that the above-referenced section of Xia (3.5) describes retrieving instructions for a trace that has already been recognized as in progress, and is not searching for the start of a trace. **Applicants respectfully disagree, and also note that even if this were true, Xia does not teach the above-referenced limitation of the claimed invention.**

The cited passage states, "In the simulator, we first check the trace table, if get instructions from trace cache, then decode them one by one; otherwise, try to get instructions from instruction cache. Although in real hardware, these two actions should be done in parallel, but it is not necessary for simulation." If, as the Examiner contends, this passage refers to the fetching of instructions from a previously stored trace, then his statement, "Examiner finds it hard to believe that this would lead one of ordinary skill in the art to check a trace cache for the start of a trace, if it is guaranteed that the instruction cannot possibly be the start of the trace line" is not applicable to the arguments made above. This passage would have nothing to do with checking for the start of a trace line, but with fetching other instructions in the trace line. **There is nothing in Xia that describes how, or more importantly when, the trace cache is checked for the start of a trace line**, as the Examiner suggest, or when the system begins a search of the trace cache for any reason.

The only clues as to when or how this may take place in Xia are found in the passage cited above (which indicates that, at least for some cases, either the trace cache is searched before the instruction cache is searched, or the trace cache is searched in parallel with the instruction cache), and section 4.4. Section 4.4 includes the following bullet point: "The trace line can only start from predictable branches rather than common instructions. **Thus the trace cache hit rate cannot be high. We cannot expect the low hit rate (2%-4.5%)** gives significant performance improvement" (emphasis added). This section clearly indicates that the cache is checked for a hit for significantly more instructions than those that could constitute the start of a trace line (hence the low hit rate). This would lead one of ordinary skill in the art to conclude that Applicants'

previous assumption based on the reading of section 3.5 is correct (i.e., that the trace cache is checked on every cycle), or at least that it is searched more often than *in response to a branch target address being generated,* as recited in claim 32, since the traces begin on branches. Applicants also point out that Xia teaches that trace lines may begin from predictable branches, but Xia also describes, "Another option is to start from any common instructions," and "Or we can try the third scheme: the valid trace must contain at least one branch or cross a cache block boundary" (see section 5.5). In these alternate embodiments, the trace cache would also need to be searched for the start of a trace in more cases than allowed by the limitations recited in claim 32. **Applicants also note that the descriptions in Xia primarily refer to methods involving how and when to build traces, not how or when to fetch them.** Therefore, Xia clearly does not overcome the deficiencies of the other cited references in teaching the above-referenced limitation of claim 32.

The Examiner's statement, "The advantages that Examiner indicated… are obvious advantages one of ordinary skill in the art would recognize, searching a cache takes time and energy, and if there is no possible way that searching a cache can benefit the user, there is absolutely no reason to do so" contradict the teachings of his own references (Rotenberg and Xia), which search the cache for a hit (although not necessarily a trace start) on every cycle. **This is clearly in direct conflict with the limitations recited claim 32.**

For at least the reasons above, the rejection of claim 32 is unsupported by the cited art, and removal thereof is respectfully requested.

In regard to the rejections under both § 102(b) and § 103(a), Applicants assert that numerous ones of the dependent claims recite further distinctions over the cited art. Applicants traverse the rejection of these claims for at least the reasons given above in regard to the claims from which they depend. However, since the rejections have been shown to be unsupported for the independent claims, a further discussion of the

dependent claims is not necessary at this time. Applicants reserve the right to present additional arguments.

CONCLUSION

Applicants submit the application is in condition for allowance, and an early notice to that effect is requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5500-88700/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant(s)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: September 18, 2007